METHOD AND SYSTEM FOR INTEGRATION OF SOFTWARE APPLICATIONS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to methods and systems for integrating software applications.

Description of Related Art

Over the past several decades, software applications have been developed as solutions in a wide variety of problem domains, for example, financial accounting, inventory control, word processing, electronic mail, and customer order entry, to name very few. With limited exception, these applications are stand-alone and offer *ad hoc* external interfaces.

Some product families, such as Microsoft Office, do offer integrated suites of software applications. However, these suites can tie a user to applications within the product family, a disadvantage where applications outside the product family might provide capabilities more suited to the user's requirements. In addition, such suites often address only a limited problem domain; a disadvantage for a user wanting to integrate solutions from outside the problem domain addressed by the suite.

Typically, the user of more than one stand-alone system or limited suite has few options for integration. One option is custom integration, typically using a brute force data-to-data converter that becomes useless when one of the applications changes. Another option is manual transcription of data between applications. However, manual transcription is subject to higher error rates and exhibits significantly lower throughput in comparison to automated methods.

Paradigms to characterize several common problem domains addressed by software systems have matured to the point where archetypal interfaces can be determined for solutions in those domains. One such problem domain includes financial accounting and inventory control.

BRIEF SUMMARY OF THE INVENTION

The present invention comprises an environment for integration of disparate software applications through exposing the functionality of subject applications in a manner that presents a predictable stable interface across an entire family of solutions.

In addition to the direct benefits offered by facilitating the exchange of data between software applications, the present invention offers an opportunity to collect operational, statistical, and analytical data of great potential value to users.

This is accomplished by means of a method for procedure normalization applied to solutions addressing a problem domain, the method comprising: identifying a union of the interfaces to a subset of the solutions; creating a generic description of that subset in a using a common descriptive meta-language; and creating solution-specific code to translate the operations and data of each interface between the each solution and the generic description.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic illustration of an integration platform architecture in accordance with the present invention;

Figure 2 is a schematic illustration of the relationship between a digital signature and its components in accordance with the present invention;

Figure 3 is an illustration of the components of a connector in accordance with the present invention;

Figure 4 is an illustration showing the relationship between digital signature, XML documentation, and connectors in accordance with the present invention;

Figure 5 illustrates addition of vendor identification through use of a client code generation tool in accordance with the present invention;

Figure 6 illustrates the logical flow of online analysis server functionality in accordance with the present invention; and

Figure 7 illustrates an integration platform architecture in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present method and system for integrating software applications has several preferred aspects including (1) a platform-independent, distributed, multi-application procedure normalization; (2) digital signature for custom connector modules; (3) a connector creation tool; (4) a client code generation tool; (5) online integration analysis; and (6) an integration knowledge management architecture.

With regard to one preferred embodiment of the present invention, the platform-independent, distributed, multi-application procedure normalization, given a problem domain (for example, accounting and inventory control for mid-size businesses), for which there exist multiple solutions (for example, software applications such as QuickBooks™) it is often desirable to provide a common interface through which most, if not all, solutions to that problem

domain may communicate. For instance, a useful tool/technology for connecting a web interface to any one of a set of legacy software applications would be one that provided consistent implementation rules and procedures across the various legacy solutions. The present invention supplies that functionality: it provides a technology-neutral representation of a given problem domain in the form of a specification, pairs it with specific programs that link the specification to application-specific technology, and facilitates the interaction of the two with an infrastructure for publishing those programmatic solutions across a platform-independent, distributed architecture.

In implementing the procedure normalization, the system and method of the present invention preferably employs a series of steps, including identification of a logical problem domain , identification of a useful union of the set of existing or proposed solutions to the specified problem domain, and the use of tagged data format technologies to create a generic description of the interfaces and data.

With regard to the identification of a logical problem domain, in this step a logical problem domain is identified. This domain can be of any scope from narrow ("Function to return IP address of physical computer system") to very broad ("Core Financial System").

Concerning the identification of a useful union of the set of existing or proposed solutions to the specified problem domain, for example, the archetypal set of interface function, procedure normalization is based on comparing existing or pending solutions to a particular problem domain and finding the common functions (interfaces) and data that exist in those solutions. Therefore, it is useful to understand a meaningful sample of solutions in a particular domain, such that the common functions and data identified are likely to exist in a broad range of similar solutions. The resulting function/data set is the one that is likely to be useful by a wide audience.

In using tagged data format, for example, XML, technologies to create a generic description of the interfaces and data, the interfaces and data of a particular problem domain which are common across specific technological solutions to that problem may be appropriately identified and described by employing a tagged data format specification (for example, XML Document Type Definition (DTD) or XML Schema Definition (XSD)).

Another preferred aspect of the present invention is connector creation. A "connector" describes solution-specific code necessary for implementing the translation and application of the specification-constrained tagged data format document representing the existing or pending solution.

In order to facilitate the creation of connectors, embodiments of the present invention provide specific tools to generate the framework of the connector, allowing the developer of the connector to focus on the task of authoring the integration-specific code. Once completed, the connector is then optionally "signed" using the digital signature tool, which is discussed later in this document.

Another preferred aspect of the present invention is registration of connector with a "Listener." A "Listener" is a program that waits for outside requests for functionality provided by connectors. In preferred embodiments of the present invention, Listeners are written in Java. The Listener can manage any number of procedure normalization specifications (problem domain specifications), and publishes the availability of these to the Server. The relationship of a procedure normalization specification to a connector providing the physical implementation of the business process is managed by the Listener and is hidden from the user. The Listener implements any number of particular communications transport, for example, RMI and SOAP, among others, (singularly or concurrently) to facilitate transmission of data across external

communications media (Internet and WAP, among others). Additionally, the Listener includes handlers for fault tolerance, to provide a stable environment in the event that a connector is not available to handle a particular outside request or a physical infrastructure fault prevents the processing of a request.

Registration of the connector with the Listener is a static process that is accomplished through editing a registration file. This registration file identifies the specification (for example, DTD, XSD) that implements the procedure-normalized specification, and the specific connector module that provides the programmatically implemented functionality. Embodiments of the present invention provide several mechanisms for invocation of connectors implemented with differing technologies (Java, Dynamic Link Libraries, Microsoft Distributed COM Objects, CORBA Objects, and Command Line Parameter Parsing, among others). The implementation code that is paired with the specification is typically supplied as a Java class. This allows the connector to be identified, loaded, and instantiated at runtime ("late binding"). If a technology other than Java is required to provide the integration functionality, this Java class passes control to the external module using the most appropriate facilitation technology. As implied, this means that multiple units can make up a connector, each providing a portion of the integration functionality.

The registration process is called a "static process" simply because the connector cannot initiate the registration with the Listener. However, a Listener can be instructed through other mechanisms (API Instruction and polling mechanism, among others) to refresh the connectors that it is publishing.

Another preferred aspect of the present invention is registration of the Listener with a Server. The "Server" is a program, written, for this embodiment, in Java, which waits for

outside requests for particular functionality exposed by various Listeners within the system. The Server manages the registration and security policies of the Listeners as well as the organization and access to the interfaces and data they publish.

The registration of the Listener with the Server is a dynamic process that identifies the Listener as a valid destination for the requests for functionality that originate from the Adapter and are brokered by the Server. Specifically, when the Server is started, it receives a list of listeners to brokerage services for (this list originates from a locally persisted data source) and contacts those listeners. If the communication with the Listener is successful, the Listener sends the Server a list of the procedure normalization specifications that the Listener supports. Furthermore, the Server establishes a relationship with the Listener whereby if the communication mechanism between the two becomes unavailable, both entities become aware of the condition in a timely manner and can take action as appropriate. Once the registration process is complete, the Server is able to route functionality requests to the appropriate Listener on behalf of the client. Should a condition that is interrupting communication between the Listener and Server rectify itself (for example, a network going down and coming back up), the technology is implemented that allows communication between the two entities to resume automatically. This same technology allows the registration process to be performed at any time, allowing the Server to dynamically update its configuration in the event that the Listener has been configured with new functionality, providing that new functionality to outside applications connecting to the Server via an Adapter (described below).

Still another preferred aspect of the present invention is the use of an Adapter to access the Server and exposed functionality. An "Adapter" is a programming object that allows various technologies simplified access to the interfaces and data exposed in all areas of the architecture

of the present invention. The Adapter provides programmatic code that simplifies the parsing, storing, and retrieval of elements and attributes against a tagged data format document, for example, XML-formatted document, by shielding the low-level mechanics of these actions from the user. Embodiments of the present invention implement the Adapter in Java and provide access to the Adapter via various industry-accepted interfaces, including C libraries and COM.

In the same way that a Listener can connect to a Server via different transport mechanisms (RMI and SOAP, among others), the Adapter can also communicate with the Server by employing similar options. The choice of Adapter transport technology is independent with respect to the choice of Listener transport technology.

Additionally, the Adapter preferably performs high-level document validation; that is, it ensures that the data supplied by the user conforms to the rules and specifications implemented by, for example, the DTD or XSD specification, to ensure that the request is in the form of a compliant, well-formed document. Thus, before any application specific code is executed, high-level errors are already eliminated (at the originating source) by the architecture of the present invention.

As shown in the illustrative embodiment depicted in Figure 1, a given application accesses a normalized exposure of backend problem domain functionality via the following logical flow:

An application 1 uses an Adapter 2 to connect to a particular Server 3. The Adapter 2 requests the initiation of an operation from the Server 3, accomplished by supplying the Server 3 with the name of a Listener 4 and the name of a procedure normalization specification (DTD/XSD) 5 within the Connector 6 that the Listener supports. The procedure normalization specification (DTD/XSD) 5 provides the defined interface and data for the backend system. The

Server 3 resolves the identity of the Listener 4 by the supplied name, requests the specification of the indicated process, and returns the document to the Adapter 2.

All communications between components occurs by means of XML documents 7. Utilizing the Adapter's simple API, the Adapter 2 constructs an XML document 7 according to the specification supplied by the Listener 4. The Adapter 2 transparently manages the task of constructing the document by shielding the user from the intricacies of working directly with programmatic XML tools.

Immediately before submitting the completed request, the Adapter 2 first validates the constructed document against the specification and determines whether the document is well-formed and compliant with the particular document definition.

The XML 7 document is submitted to the Server 3, which determines the user's eligibility to access the exposed functionality and logs the activity accordingly.

The XML document 7 is forwarded to the Listener 4, which calls the appropriate invocation mechanism to pass the information to the Connector 6 corresponding to the request.

The Connector 6 executes the appropriate system-dependent code and constructs an XML, response document 7 containing either return codes or data, depending upon the nature of the particular request.

The XML response document 7 is routed back through the Listener 4 and Server 3 (where it is again logged) and finally back to the Adapter 2.

The application 1 can then use the functions within the Adaptor 2 to inspect the contents of the returned XML response document 7 and take appropriate action based on those results.

With regard to the digital signature for custom connector modules, a preferred embodiment of the present invention provides a digital signature technology that allows a

certified developer to "sign" their authored source module and associate that signature with a tagged data format, for example, XML, specification that identifies the interfaces and data exposed by the module for the purpose of validating that a particular integration module was authored by a specific, certified developer.

This preferred embodiment has several aspects, including a binary hashing algorithm, certification, custom identifier generation, and signature interpretation.

In the binary hashing algorithm, a unique "hashed" number may be generated by inspecting the binary footprint of a particular executable code module by utilizing numerical algorithms. In certification a certification authority assigns unique certification numbers to each developer who completes training and other requirements for the Certified Developers Program. With custom identifier generation, each "signed" solution will be given a custom identifier that indicates the link between a particular specification and a specific version of an executable module by combining the numerical results of the Binary Hashing Algorithm with the Certification number. In signature interpretation, the Listener has the ability to examine the Connector and specification components of an integration solution and validate their authenticity at run-time. This information is then made available to the Server during the registration process.

As shown in the illustrative embodiment of Figure 2, three components are provided to the binary hashing algorithm 10. These three preferred components are the tagged data format (for example, XML) specification 11, the executable file (or files) 12 that contains the programmatic code implemented the interfaces and data specified in the specification, for example, the DTD/XSD, and the certified ID 13 of the developer who created the executable file.

These three preferred components are processed by the binary hashing algorithm to produce a custom identifier or digital signature 14, which is then programmatically associated with the tagged data format specification and the specified executable. This can later be used to confirm that the executables specified by the specification and the specification itself, have remained unchanged since the time that the custom solution was originally written and signed.

With regard to the connector creation tool, a "connector" describes solution-specific code necessary for implementing the translation of a particular specification within a particular existing or pending solution. Furthermore, in the case of a preferred connector, there are preferably the additional components of a tagged data format document describing the connector functionality and an digital signature certifying the authorship of the connector. In Figure 3 is depicted a particularly preferred construction of a connector, comprising XML documentation 20, an API specific code 21, an XML DTD/XSD functional description 22, and an optional digital signature 23. The connection creation tool is an automated software program that facilitates the simple and consistent creation of connector objects.

The tagged data format specification connection creation tool is used to create the functional description component of the connector. It permits the author of a connector to specify functional parameters, data structures, and other fixed-type data elements. Then, the connection creation tool creates a tagged data format specification that accurately describes those elements. This permits a programming author personally unfamiliar with the syntax of tagged data format specification documents to, nonetheless, create those documents corresponding to the functionality of specific code.

As the author of a connector adds interfaces, data structures, and other logical objects to the specification that describes the functionality of the connector; the tagged data format

documentation creator provides interactive creation of documentation that is linked directly to the contents of the specification. This allows the programming author personally unfamiliar with the creation of documents in the tagged data format (for example, XML) the ability to quickly create the necessary documentation for their Connector.

The connector creator user interface provides a common platform for performing the functions necessary to create a connector and provides the author a single place to create the specification describing the functionality of their Connector and to create the tagged data format document that provides the documentation for how that specification should be used, and provides a simple interface for specifying the API-specific executable and creating a custom digital signature.

As illustrated by the embodiment shown in Figure 4, the connector creation tool is used in iterative steps to define functionality within the connector and document that functionality in a tagged data format document. Preferably, some external programming is necessary to provide the functionality in an executable program module. Following these iterative steps, a second process is provided that permits an author to create a custom digital signature indicating the completion status of the connector and automatic insertion of a new signature into the specification, for example, associated with the solution. A graphical illustration of this process is provided in Figure 4.

The client code generation tool is preferably provided for the purposes of generating source code for various development platforms so as to simplify the rapid development of applications designed to run within in environment of the preferred embodiments of the present invention. The client code generation tool preferably provides a simple drop down list with which to browse available and accessible Listeners within the system. A preferred operation

browser provides a hierarchical view of the specifications that are exposed by the Listener selected in the Listener browser. The operation browser permits the user to navigate through the structure of any specification in order to inspect that interface.

In a particularly preferred embodiment of the present invention, an interactive help window dynamically displays help text associated with any operation (or other item) selected in the operation browser.

In addition to dynamically displayed help information, the client code generation tool also provides code samples of how to implement particular functionality using various Adapter technologies. Multi-language code preview tabs display these code samples in a variety of implementation languages. Additionally, at this point, the client code generation tool preferably has copied this code to the system clipboard so that it may be easily pasted into a third-party development environment.

An example of adding a Vendor ID to a particular integration application written in C++ is illustrated in Figure 5. The multi-language capabilities are evident in the tabbed dialog section at the bottom of the client code generation tool main window. By selecting the "Java" tab, for example, the code sample shown would change to an appropriate syntax for a Java compiler.

With regard to the online integration analysis, the Server component of the architecture preferably logs various transactions that occur within the integration system. These include registration of Listeners, published operations of the Listeners, transactions from Adapters, error conditions, and other data. The online integration analysis component allows a user to access a web page on the Internet and view reports on the details of a specific integration solution.

Preferred embodiments of the present invention provide an online analysis server on the Internet that publishes a set of web pages designed to assist users in analyzing their integration solution.

Figure 6 illustrates the preferred steps for viewing an online integration analysis of a specific integration solution: In this figure, the online analysis server 1 provides a web page for access to analysis reports 5. The user preferably connects to the analysis server web page and specifies the Internet IP address of their integration server 4. Additionally, user can select a specific report to view. The online analysis server 1uses SOAP 3 to connect to integration server 4 and requests data necessary for the report. The integration server 4 retrieves relevant data from its logging database and returns it via XML response document in a SOAP packet. The online analysis server uses XML response documents to construct HTML report of data and publishes it to the Internet. The user is, thus, able to see the report without the overhead of report generation tool or local web server.

Concerning integration knowledge management architecture, the unique combination of the technologies provided by embodiments of the present invention combine to create a comprehensive solution to the problem of managing integration knowledge. This innovative system allows an enterprise an unprecedented ability to distribute and actively employ integration technologies regardless of geographic boundary.

The integration knowledge management architecture provides a common framework for the management of integration knowledge and the ability to communicate both that knowledge and connectivity to its implementation throughout the enterprise. The integration knowledge management architecture consists of various inter-dependent technology modules that provide a powerful, structured environment for performing the following four integration knowledge tasks:

The author component of the connector creation tool provides a simple interface for IT development staff to create custom connectors to back-end systems (legacy systems and middle-tier financial systems, or other applications, among others). By providing an IT professional with a simple mechanism to logically bind an authored executable function with a simple tagged data format (for example, XML) interface, the present invention provides developers a simple, open, and flexible connection to specific back-end areas of expertise. The connector creation tool provides an interactive environment for creating context sensitive help documents to provide additional integration information.

With the preferred publish feature of the present invention, the connector creation tool provides a simple mechanism for a developer to digitally create a signature, specifying ownership of his/her work. This signature technology is important for verification of authorship, an integral part of the knowledge management problem. Once signed with the digital signature, the author may register a created component with the Server. This process of publishing the connectors allows for distribution of integration knowledge throughout the enterprise. In addition, the CTO or IT manager can use the security facilities within the Server to logically create integration knowledge views, specific groupings of published connectors with secured access by specified groups of users.

Another preferred embodiment of the present invention is an execute function within the integration knowledge management architecture, providing the ability to quickly execute any integration functionality exposed within the system of the present invention. By utilizing the code generation tool, a developer, not only has access to the context-sensitive information provided by the connector; but also preferably has the ability to immediately access the implementation of that connector from any location that has access to the Internet. This

powerful component allows a security-based deployment of individual "components" throughout the enterprise.

Still another preferred embodiment of the present invention is an analysis function, providing a reporting environment that provides appropriate members of the enterprise access to both the availability and use of individual integration connectors. This reporting tool preferably provides live access to executing servers and listeners and can give statistical and analytical information on what functionality is available to the enterprise, the portability of those functions, the ownership of those functions, and empirical data about their frequency of operation and effectiveness.

Figure 7 illustrates a life-cycle solution for a preferred embodiment of enterprise knowledge management comprising, preferably, the steps of authorship of connector, publication of connector, execution of functionality, and analysis of integration.

For example, with regard to authorship of connector, an IT professional preferably creates a connector for an internal database called "<Employee Travel Profile>", which provides information about where individual employees travel on business-related trips. This information includes historical information such as frequent-flyer numbers and room preferences. Using the connector creation tool, the developer documents all of the data necessary to do a lookup and the specific information of the data that will be returned.

Concerning the publication of connector, in this example, the employee uses the connector creation tool to apply his/her digital signature to the completed component, at which time he/she publishes the component to the internal Oracle Listener connected to the relevant system. The Listener then updates the Server with the new information regarding the existence of this new connector. The integration knowledge management administrator then preferably

determines the relevance of this connector to the enterprise and places it into an appropriate folder for viewing.

Regarding the execution of functionality, in this example, a developer within the enterprise creates an Intranet to allow employees to quickly enter travel requests with the Human Resources scheduling department. Knowing that another system exists within the enterprise for tracking similar data, the developer queries the integration knowledge management server for similar functionality. Upon inspecting the entry for <Employee Travel Profile> and its constituent documentation, the developer determines that they can leverage that functionality to greatly reduce the amount of information entered at the time of a travel request. By using the client code generation tool, one is provided with a web-based interface that permits an employee to simply enter his/her employee identification number and password. Then, the backend Oracle system is queried using the interface described by the knowledge management architecture. This allows the application immediate access to the various data within that system, which is then populated into the application by means of code generated by the client code generation tool.

Concerning analysis of integration, in this example, an IT manager contemplating eliminating the Oracle database can identify a dependency on the information it contains. Additionally, he/she can statistically identify the amount of requests that data serves, the source of those requests, and the type of data requested and returned. Finally, if he/she chooses to replace the database with another system, he/she knows what functionality needs replication in the new solution.

Another aspect of the present invention is web interaction. Organizations often deploy integration solutions that tie two or more systems together via programmatic methods. But this path to integration ignores the opportunity to expose the organization's business processes to

employees, customers and partners in ways that allow them to do business via web services. The present invention preferably permits the rapid combination and management of the three core components of an integration solution: data, business processes, and the knowledge of how the components all works together.

This preferred embodiment of the present invention is an interaction portal, composed of a web-based user interface, an interaction registry, and components for development of services, client application connections, and web access. These components enable users to create a new interaction service, translate between two different services, gain access to a database, automatically generate web pages, create automatic client-side connections, generate Java, C++, Visual Basic, and/or C# code for service creation, deliver asynchronous COM/CORBA/EJB integration, and deploy the solution and tools simply and quickly (web-based, on demand).

The interaction portal of the present invention speeds up development, reduces risk, increases the ability to maintain the solution, and allows business experts to focus their skills on the problem where needed. The tools, technology, and methodology of the present invention insulate development efforts from the wide and ever-changing array of integration platforms and technologies available today, providing an unified and vastly simplified development process. The solution of the present invention is simplified, because developers do not need knowledge of complex concepts such as distributed object protocols or data encryption, because the present invention handles these issues in a number of different languages (Java, Visual Basic, C/C++/C#, among others). The solution of the present invention is unified, because the code is integrated with a variety of application servers and service-based middleware products, including the integration portal.

In a preferred embodiment of the present invention, interactions with the interaction portal are carried out by means a predefined operation previously identified. For example, in the case of an e-mail demonstration, one identifies the operation in the body of the email message, then supplies the specific details in a message attachment. To complete the email example, one specifies the details of the requested operation. The operation itself was defined in the body of the email message. Now, a simple text file attachment will provide the details necessary to fulfill that request. All of this information can be (and, in this case, has been) provided without any programming or direct connection to a web server. This kind of interaction can be very helpful for clients, employees, or partners who are not directly "wired" to an enterprise.

Preferably, when one sends the email message, one receives two email messages in return. The first is an acknowledgement of your request, the second will contain the results.

Thus, the interaction portal of the present invention requires only two pieces of information: What do you want to do? and What are the specifics of your request? The present invention provides a variety of ways to submit these two pieces of data to the interaction portal, including the local file system, Internet FTP, and SMTP.

The present invention provides a means by which people can participate in the organization's business processes via email, or other standard software applications. This demonstration of human interaction demonstrates the ease by which an organization can provide access to key business processes, even when the interactions are with employees, customers and partners who aren't "wired" to the enterprise.

In addition, the systems interaction aspect of the present invention is also a feature. The true value of integration lies within an organization's ability to improve competitiveness through information and process sharing. Rather than focusing on exposing data at a technical level, the

present invention helps organizations communicate at a business level -- insulating business development efforts and processes from ever-changing technology and integration solutions.

The present invention provides the ability to programmatically access a business process through the interaction portal. Data is submitted to the portal by means of a client connector. The interaction portal preferably generates HTML forms for performing business interactions. This technology permits company to quickly deploy solutions via its corporate intranet or the Internet. Thus, anyone with a web browser can access the data, anywhere, at anytime.

For example, a single interaction "provider" can be implemented. This provider receives requests in an XML format and executes operations against a database according to the contents of those requests. The results of the operations are then placed back into XML format and returned to the portal. This architecture is powerful because it completely isolates the business functionality and implementation details from the endpoints of the transaction. In short, that frees a company to select best-of-breed solutions without worrying about the impact on the rest of the IT infrastructure. It also allows technical personnel to focus on core business needs, instead of dealing the with communications and messaging architecture that ties business systems together.

Various preferred embodiments of the invention have been described in fulfillment of the various objects of the invention. It should be recognized that these embodiments are merely illustrative of the principles of the invention. Numerous modifications and adaptations thereof will be readily apparent to those skilled in the art without departing from the spirit and scope of the present invention.